Chapter 5

Planning and Scheduling

Excessive or irrational schedules are perhaps the single most destructive influence in all of software engineering! — Casper Jones

5.1 The Basic Principles of Project Planning

Let us recap briefly what has been covered so far.

- **Processes** — We saw how processes are used to manage the development of large or complex software systems and to ensure that all of the key analysis and design steps are taken so that the software system developed is actually fit for purpose.
- **People and Teams** — People working in teams must work together using a process to create the software. We saw what motivates people and how to influence them, coerce them when necessary and how to structure them into the right sort of teams to work on a project.

We now need to combine these ingredients — the project, the processes and the people — to create a *project plan*. The project plan must:

1. define the tasks that need to be carried out as part of the processes that are being followed;
2. the duration and dependencies for each task;
3. the people and physical resources required by each task; and
4. the milestones or goals of each task.

The big danger when producing a project plan is to make an infeasible plan. In fact the main reason that many projects go over-time and over-budget are related to the project plan [Pre09]:

- **Unrealistic deadlines** typically established by someone outside of the project team and forcing the schedule on the project team.
- **Changing requirements** that are not reflected in the project plan.
- **Honest underestimates of effort** and the amount of resources required to complete the project.
- **Risks** that were either predictable or unpredictable at the start of the project but not accounted for in the project plan.
- **Technical difficulties** that could not have been foreseen at the start of the project.

- **Human difficulties** that could not have been foreseen at the start of the project.
- **Failure to see slippage** and for the project management to see that the project is falling behind schedule.
- **Mis-communication** between project staff causing delays.

The idea is to analyse the people, projects and processes early in the planning phases of a project to arrive at more realistic schedules. If the estimates in the project plan result in a schedule that exceeds the given deadline then there is a problem. The advice is always to discuss the schedule with the client and to give your reasoning for the estimates in the schedule. If the client does not accept a modification to the schedule — and this can happen for a number of reasons not always to do with the client — then iterative or incremental models with management structures that support shorter deadlines and phased delivery are to be preferred.

Basic Concepts

There are basic principles to project planning just as there are basic principles to most things in software engineering. For project planning they are the following [Pre09].

1. **Compartmentalise** — The aim here is to decompose the project until the activities, actions and tasks are manageable. The basis for accomplishing this decomposition is typically to decompose both the product and the process.
2. **Interdependency** — The interdependencies between activities, tasks and actions need to be determined. Dependencies can take a number of forms. Tasks may be sequentially dependent on each other, for example, when one task must use the work product produced by another task, or when the purpose of one task is to establish some ground conditions for another task. One the other hand, if there are no dependencies between tasks then they may occur in parallel.
3. **Time Allocation** — This is one of the harder tasks in project planning. An estimate of the number of resources or effort — for example, person days — must be included for each task in the plan. Then, the start date and completion date, based on the interdependencies, and whether or not the work will be completed on a part-time or full-time basis needs to be estimated.
4. **Effort Validation** — The number of people and resources on a project are typically finite. Validation is the management activity of going through the project plan to ensure that only the available number of resources are allocated at any given time.
5. **Defined Responsibilities** — Every scheduled task should be assigned to a specific team member.
6. **Defined Outcomes** — Every scheduled task should have a well defined outcome. The outcomes for software engineering tasks are typically work products or parts of work products.
7. **Defined Milestones** — Every task or group of tasks is associated with a defined milestone. Milestones are accomplished in the project plan when a work-product or group of work-products are accepted for further development.

People and Effort

A common myth is that if a project is behind schedule then the schedule can be shortened simply by adding more resources. This is not true. Brooks [Bro95], in his seminal paper, *The Mythical Man-Month*, makes the following observation:

> “Men and months are interchangeable commodities only when a task can be partitioned among many workers with no communication among them. This is true of reaping wheat or picking cotton; it is not even approximately true of systems programming.”

Brooks contends that estimating using *man-months* — that is, the number of months it would take a person to complete a task — can be misleading. A task that takes ten months for a single person is unlikely to take one month for ten people.

His research demonstrates that adding people to a project when it is already behind schedule can cause more disruption because of unfamiliarity with the system and even more delays. People who are added must learn the system. This often takes people off the project to help the new people, and thus the project falls even further behind. Furthermore, as new people are added the number of communication channels increases and so may slow the work down even more.

An empirical relationship between effort, measured in terms of person-months, and development time has been established over time. The relationship is sketched in Figure 5.1, and is known as the *Putnam-Norden-Rayleigh curve*; named after the three people that contributed to it.

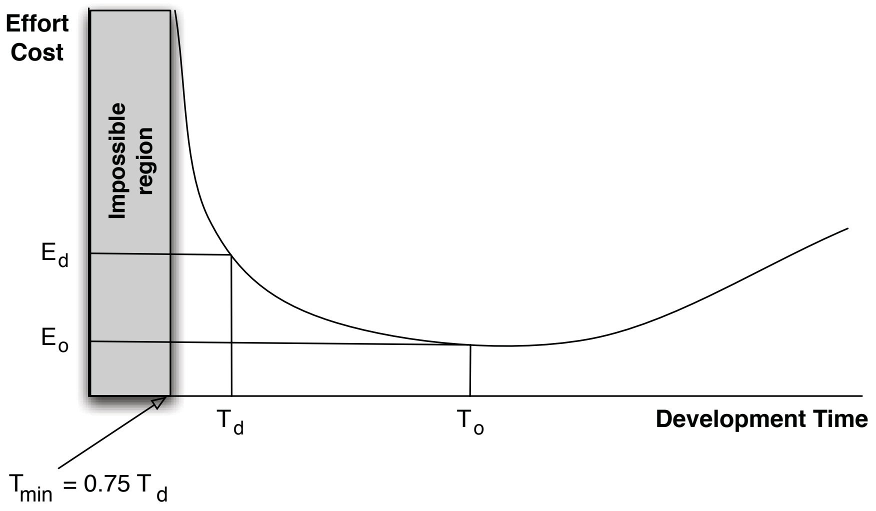


Figure 5.1: The relationship between effort and delivery time. The formula relating effort and delivery time is $\mathrm{E_o} = \mathrm{m}(\mathrm{T_d^4}/\mathrm{T_a^4})$, where $\mathrm{E_o}$ is the effort in person months, $\mathrm{T_d}$ is the nominal delivery time as estimated by the schedule, $\mathrm{T_o}$ is the optimal delivery time in terms of cost and $\mathrm{T_a}$ is the actual delivery time.

The development time with the *least cost* is denoted by $\mathrm{T_o}$. The development time with least cost is not often the shortest development time. As we move left from $\mathrm{T_o}$ then we are effectively accelerating the project and the corresponding effort rises non-linearly.

As an example, suppose that we have estimated a nominal delivery time of $\mathrm{T_d}$ given the project resources with the corresponding effort $\mathrm{E_d}$. If we wish to accelerate the project then the curve indicates that we cannot accelerate it much beyond $0.75\mathrm{T_d}$. Shortening the delivery time further than $0.75\mathrm{T_d}$ results in a very high risk of failure. Of course lowering the cost to the right of $\mathrm{T_o}$ needs to be carefully weighed against business opportunities and business cost.

Work Breakdown

The software development lifecycle model chosen for a project is often the first step in developing the project plan. For example, suppose that we chose a waterfall process for our project. The phases in the waterfall naturally lead to a set of tasks.

1. **Concept**
2. **Requirements**
3. **Design**

4. Implementation

5. Acceptance Testing

The principle of compartmentalisation requires that we break this high-level set of tasks down into more manageable tasks. The task breakdown can be done using a diagram and this results in a tree-like structure as in Figure 5.2, called the *work breakdown structure*.

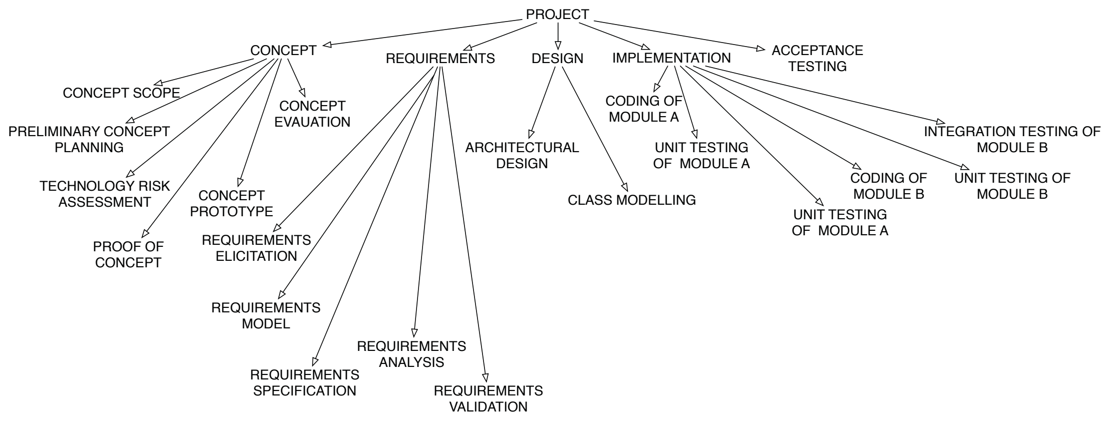


Figure 5.2: An example of a work breakdown structure.

The same work breakdown structure can be given in tabular form as shown in Fig. 5.3.

A popular rule that is used in defining the work breakdown structure for the project is the *100% rule*.

Definition 5.1. *100% rule*

The 100% rule states that the work breakdown structure includes 100% of the work defined by the project scope and captures all deliverables — internal, external, and interim — in terms of the work to be completed, including all project management.

Note that this is not always possible in the early stages of the project but what this rule means in practice is that as new work-packages are identified then they must all be built into the existing work breakdown structure and into the project plan.

Dependencies

Dependencies may exist between tasks. Dependencies between tasks may exist for a variety of reasons including:

- A task relies on a work product produced by another task; for example, the design task needs a requirements specification produced by the requirements task.
- A task relies on a work product (developed by another task) to be in a specific state before it can commence; for example, requirements and design may be done in parallel but the design task relies on the shared specification to be at a *baseline*[1].
- A task needs the resources used by another task.

Dependencies force sequencing on the set of tasks: if task A depends on task B, then task B must be scheduled before task A. The result of analysing all of the dependencies is a *task network*. An example is shown in Figure 5.4.

[1]We will talk more about baselines in Chapter 6.

1. Concept

1.1 Concept Scope
1.2 Preliminary Concept Planning
1.3 Preliminary Analysis
1.3a Technology Risk Assessment
1.3b Initial Requirements
1.3c Build Configuration
1.4 Proof of Concept
1.5 Concept Prototype
1.6 Prototype Integration
1.7 Concept Evaluation

2. Requirements

2.1 Requirements Elicitation
2.2 Requirements Prototype
2.3 Requirements Analysis
2.4 Requirements Specification
2.5 Requirements Validation

3. Design

3.1 Software Architecture Design
3.2 Class Models

4. Implementation

4.1 Coding the Client
4.2 Testing the Client
4.3 Coding the Server
4.4 Testing the Server
4.5 Integration Testing of Client with Server

5. Acceptance Testing

Figure 5.3: A work break-down structure.

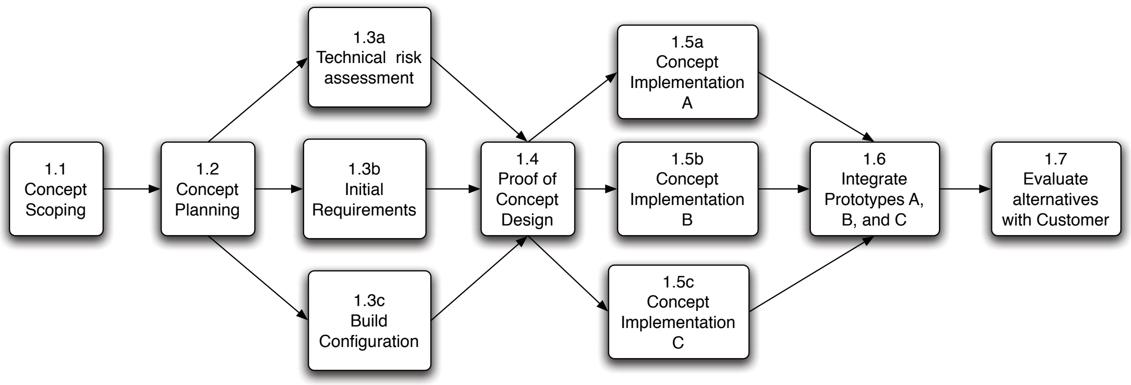

1.1 Concept Scoping
1.2 Concept Planning
1.3a Technical risk assessment
1.3b Initial Requirements
1.3c Build Configuration
1.4 Proof of Concept Design
1.5a Concept Implementation A
1.5b Concept Implementation B
1.5c Concept Implementation C
1.6 Integrate Prototypes A, B, and C
1.7 Evaluate alternatives with Customer

Figure 5.4: A task network for the concept phase of the WBS in Figure 5.2.

5.2 Working with the Project Plan

So far we have broken our project down into tasks and dependencies and have developed an activity network. The key questions are now: "*How long will the system take to develop*"? and "*How much will it cost*"? Estimates of duration and cost can be made by developing a *project schedule* in which all the tasks, their estimated duration and their estimated cost are laid out. Note that we already have part of the schedule in the tasks and dependencies in the task network. What is left however is the hard part; the time estimates and resources needed for each task.

Two kinds of graphical notations are commonly used to show the project schedule. *Activity charts*, like the one in Figure 5.4, show the dependencies between tasks and show the *critical path* for the projects. *Bar charts*, such as Gantt charts, show the schedule of tasks against calendar time. Both views of the project schedule are useful.

PERT Charts and Gantt Charts

PERT (Program Evaluation and Review Technique) charts represent the project schedule as an activity network, like the task network in Figure 5.4. The original PERT charts were designed for projects with uncertainty and so are ideal for the early stages of project planning. A Gantt chart presents the same kind of information as a PERT chart except that it shows the duration of each activity against calendar time, which PERT charts do not. Because Gantt charts are a tabular representation of the project schedule, they can be used to present a large amount of information in a relatively compact form.

Before going into detail we will need some (. . . a lot of) terminology.

- A *milestone* is an event that takes zero time. It is often used to represent the completion of an activity or the delivery of work-product.
- An *activity* is part of a project that requires resources and time.
- The *free float* or *free slack* is the amount of time that a task can be delayed without causing a delay to subsequent tasks. The *total slack* or *total float* is the amount of time that a task can be delayed without delaying project completion.
- The *critical path* is the longest possible continuous path taken from the initial event to the terminal event. It determines the total calendar time required for the project; and, therefore, any time delays along the critical path will delay the reaching of the terminal event by at least the same amount.
- A *critical activity* is an activity that has total float equal to zero. Activities with zero float are not necessarily on the critical path.

PERT Charts

PERT charts are activity networks that make estimates of project durations by decomposing the project into tasks and dependencies. Each node in the network is a model of the project task taken from the work breakdown structure. Edges in the network model dependencies between tasks.

The aim of PERT analysis is come up with a schedule for the project in order to estimate total project time and total project resources. PERT is ideal for the situations in which there is uncertainty because it makes use of (bounded) uncertainty in the duration tasks as part of the analysis. A typical analysis makes use of the following concepts and estimates:

- A *predecessor node* is a node that immediately precedes some other node without any other nodes intervening. It may be the consequence of more than one activity.
- A *successor node* is a node that immediately follows some other node without any other nodes intervening. It may be the consequence of more than one activity.
- The *optimistic time (O)* is the minimum possible time required to accomplish a task, assuming everything proceeds better than is normally expected.

- The *pessimistic time (P)* is the maximum possible time required to accomplish a task, assuming everything goes wrong (but excluding major catastrophes).
- The *most likely time (M)* is the best estimate of the time required to accomplish a task, assuming everything proceeds as normal.
- The *expected time (TE)* is the average time the task would require if the task were repeated on a number of occasions over an extended period of time. Expected time is estimated by the formula:

$$\mathrm{TE} = (\mathrm{O} + 4\mathrm{M} + \mathrm{P})/6$$

One aim of a PERT analysis is to determine the characteristics of a project plan that will let project managers do scheduling trade-offs and allow them to monitor project progress. Part of this process involves calculating the following estimates:

- Earliest start time (ES),
- Latest start time (LS),
- Earliest finish time (EF)
- Latest finish time (LF)
- Slack time

The calculation of *earliest start* times and *earliest finish* times requires a forwards pass through the network while the calculation of *latest start* and *latest finish* times requires a backwards pass through the activity network. To calculate earliest start and finish times:

Step 1 Estimate the expected time of all of the tasks in the activity network. Ideally, you will need to build up some history of projects and the times that specific tasks took to complete so that you can make better estimates.

Step 2 Start by selecting all of the activities with no dependencies. The earliest start time for each of these is *day 0*. The earliest finish time for each of these is their duration.

Step 3 Next identify all of the successor nodes to those identified in the previous step.

If a node has a single predecessor then its earliest start time is equal to the earliest finish time of its predecessor.

If a node has multiple predecessor nodes then its earliest start time is equal to the maximum earliest finish time of all of its predecessors.

Step 4 Repeat Step 3 until there are no more activities left.

There is a question of what time to use as the *duration* of a task. If we use the most likely time then the estimate may be too low if some tasks take their pessimistic time to complete. If we use pessimistic times for the duration of a task then our estimate may be too long and we may find ourselves costing the project too high. If we take only the optimistic times then we may end up under-pricing our project and so lose money. There is no right answer for this — project managers rely on their experience to determine which time to use for different projects.

The calculation of the latest start and finish times is very similar to the calculation of earliest start and finish times except that is done by a backwards pass through the network. We again use the expected time as the duration for the task.

Step 1 Start by identifying the activities with no successor nodes. The latest finish time for each of these is final day of the project. The latest start time for each of these is their latest finish time minus their duration.

Step 2 Next identify all of the predecessor nodes to those identified in the previous step.

If a node has a single successor then its latest finish time is equal to the latest start time of its successor.

If a node has multiple predecessor nodes then its latest finish time is equal to the minimum latest start time of all of its successors.

Step 3 Repeat until there are no more activities left.

The free slack is calculated as LF - EF. Typically, all of this data is represented in a single box in a PERT node as in Figure 5.5.

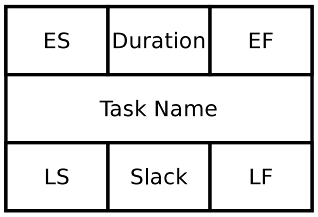


Figure 5.5: The data associated with each task in a PERT analysis modelled as a node.

Example 5.1. PERT Chart Example

Consider the tasks and dependencies from Figure 5.4. Table 5.1 shows the estimates for the tasks in this network. The corresponding PERT chart, which uses the most likely time estimates, is shown in Figure 5.6.

Task	**Dependencies**	**Most Likely Time**	**Optimistic Time**	**Pessimistic Time**	**Expected Time**
1.1 Concept Scoping		2 days	1 day	3 days	2 days
1.2 Concept Planning	1.1	2 days	1 day	3 days	2 days
1.3a Technology Risk Assessment	1.2	1 day	1 day	1 day	1 day
13b Initial Requirements	1.2	4 days	2 days	6 days	4 days
13c Configuration	1.2	2 days	1 day	3 days	2 days
1.4 Proof of Concept	1.3a, 1.3b, 1.3c	1 day	1 day	1 day	1 day
1.5a Concept Prototype A	1.4	5 days	3 days	7 days	5 days
1.5a Concept Prototype B	1.4	2 days	1 day	3 days	2 days
1.5a Concept Prototype B	1.4	3 days	1 day	5 days	3 days
1.6 Prototype Integration	1.5a, 1.5b, 1.5c	4 days	2 days	6 days	4 days
1.7 Concept Evaluation	1.6	3 days	2 day	4 days	3 days

Table 5.1: Tasks from Figure 5.4 and their durations.

Gantt Charts

Gantt charts are a common technique for representing the same information as a PERT chart. The key difference is that a Gantt chart displays tasks as a bar chart against calendar time. A common mistake is that Gantt charts are actually used to create a project design and that they define work breakdown structure. This is not the case. We have stressed that process design is about choosing a set of processes to meet a goal and the work breakdown structure defines the tasks that need to be done as part of the process. Gantt charts simply represent this design as a project schedule.

Gantt charts only focus on schedules and not on scope or cost. They can however be used show dependencies and resources as well as the progress of the project against calendar time. An example of a Gantt chart is shown in Figure 5.7.

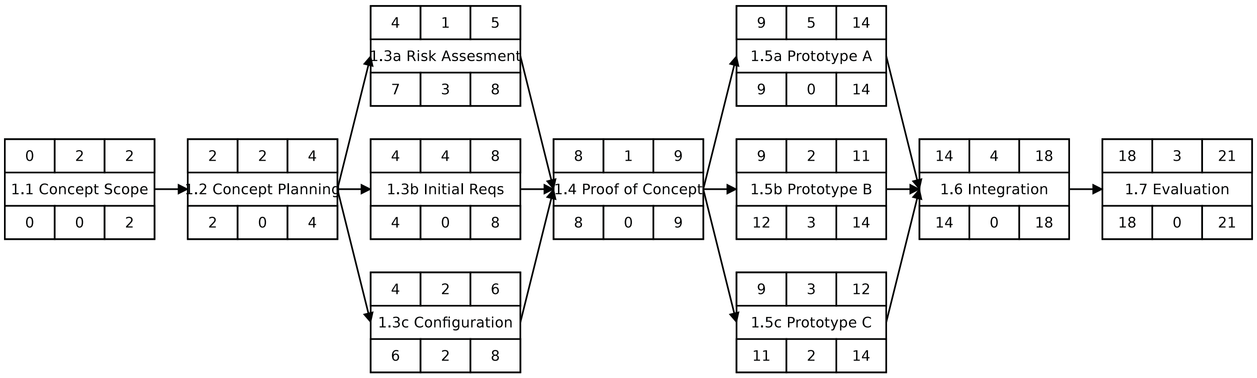
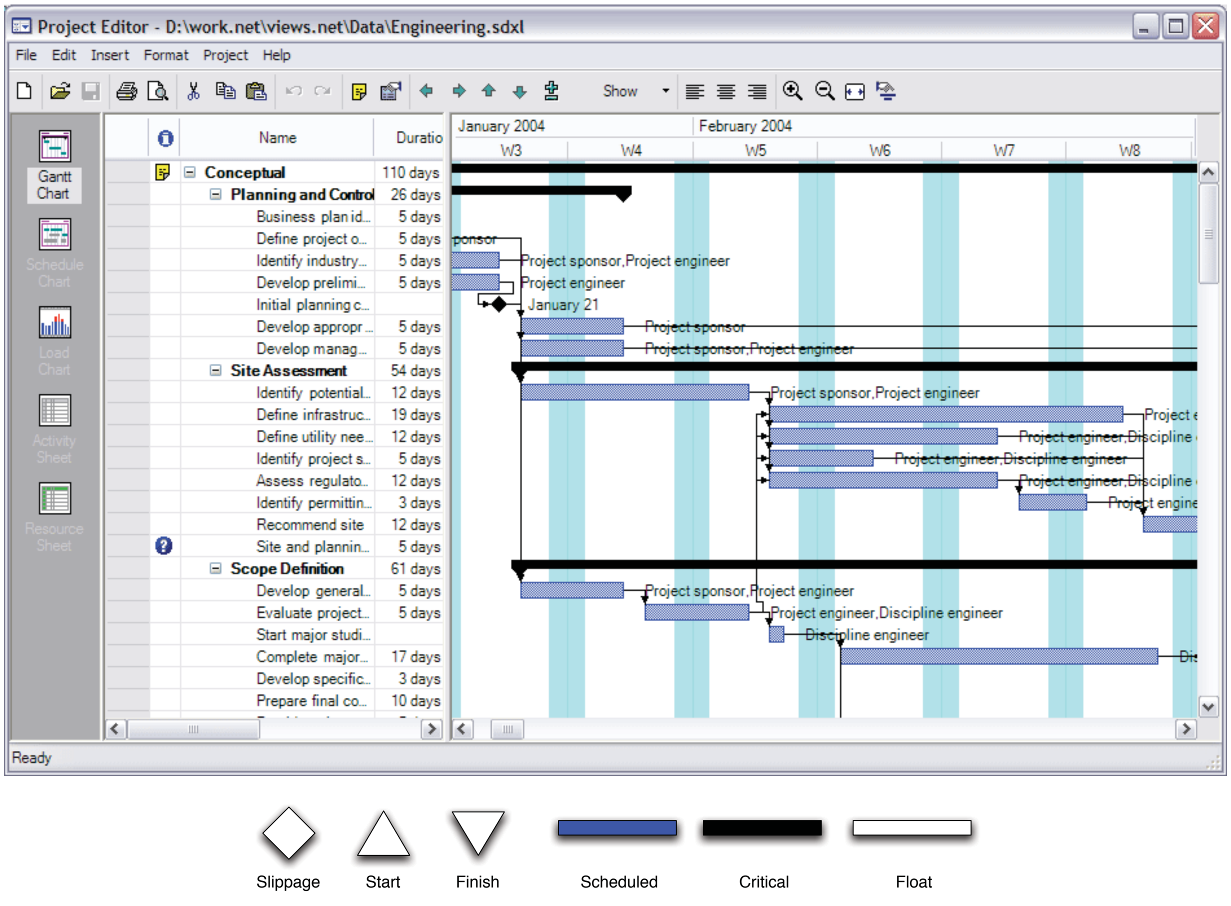


Figure 5.6: The PERT chart for the data in Table 5.1.



Project Editor - D:\work.net\views.net\Data\Engineering.sdxl
File
Edit
Insert
Format
Project
Help
Show
Gantt Chart
Schedule Chart
Load Chart
Activity Sheet
Resource Sheet
January 2004
February 2004
W3
W4
W5
W6
W7
W8
ponsor
Project sponsor,Project engineer
Project engineer
January 21
Project sponsor
Project sponsor,Project engineer
Project sponsor,Project engineer
Project e
Project engineer,Discipline
Project engineer,Discipline engineer
Project engineer,Discipline
Project engine
Project sponsor,Project engineer
Project engineer,Discipline engineer
Discipline engineer
Dis
Ready
Slippage
Start
Finish
Scheduled
Critical
Float

Name	Duratio
Conceptual	110 days
Planning and Control	26 days
Business plan id...	5 days
Define project o...	5 days
Identify industry...	5 days
Develop prelimi...	5 days
Initial planning c...	
Develop appropr...	5 days
Develop manag...	5 days
Site Assessment	54 days
Identify potential...	12 days
Define infrastruc...	19 days
Define utility nee...	12 days
Identify project s...	5 days
Assess regulato...	12 days
Identify permittin...	3 days
Recommend site	12 days
Site and plannin...	5 days
Scope Definition	61 days
Develop general...	5 days
Evaluate project...	5 days
Start major studi...	
Complete major...	17 days
Develop specific...	3 days
Prepare final co...	10 days

Figure 5.7: An example of a Gantt chart.

Critical Path Methods

A path is a sequence of nodes $a_0 a_1 \dots a_k$ in the activity network in which for each a_j, a_{j-1} is the predecessor of a_j and a_{j+1} is the successor of a_j. The total duration of a path is the sum of the durations of all of all the nodes in the path:

$$\mathrm{Time} = \sum_{i=0}^{k} \mathrm{duration}_i .$$

The *critical path* is the path with the longest duration. The overall duration of the critical path estimates the total time that project will take. All activities on the critical path have a free slack of 0. Any delay in starting or finishing an activity on the critical path will delay the total completion time of the project.

Example 5.2. Critical path

In the example in Figure 5.6 there are 9 possible paths. The longest path is the path consisting of the tasks

$$1.1,\ 1.2,\ 1.3c,\ 1.4,\ 1.5a,\ 1.6,\ 1.7$$

with a total duration of 26 days. Notice that the earliest finish times and latest start times of activities on the critical path coincide giving no free slack.

To reduce the time take for the overall project the critical path must be reduced. The idea is often to get additional resources for the activities on the critical path to either:

- remove some of the dependencies between activities on the critical path; or
- shorten the durations of the activities on the critical path — but we must be mindful of the Putnam-Norden-Rayleigh curves.

Shortening the project schedule in this way is often referred to as *crashing the project plan*.

Project Tracking and Control

The project plan is the roadmap for the project manager. If the project plan has been well analysed and well developed then the activities, resources, schedule, deliverables and milestones in the project plan will allow the project manager to track the progress of the project.

Tracking the project plan can be accomplished in a variety of ways.

- Period reviews where team members report progress.
- Evaluating the results of reviews and audits conducted as part of the software engineering process.
- Determining whether formal project milestones have been accomplished by the scheduled data.
- Comparing actual start dates with scheduled start dates.
- Meeting informally with the engineers working on the project to get their subjective assessments of progress and upcoming problems.
- Using a formal method like *earned value analysis* [Pre09].

The aim of tracking the project is to exercise control.

If things are going well then there is typically little control that needs to be exercised. If there are problems in the project then the first step is to *analyse the problem* (and avoid just reacting). After that the project manager must exercise control to reconcile the problems as quickly possible. It is not uncommon to have to find additional resources to put into the problem area or to shift staff from another area of the to the problem area — although one must remember the lessons of the Mythical Man-Month [Bro95].

Whatever the solution to the problem it usually means having to redefine the project plan.

5.3 Planning in agile development

In agile projects, planning takes on a significantly different flavour to what we have discussed so far. The most striking difference from a high-level view is that detailed planning is not undertaken until the start of an iteration.

Agile processes are designed to handle change. A prioritised list of requirements is maintained, but requirements can be added, removed, or changed at any time. The theme of agile development is to not plan to build any of these requirements until directly before you build them, because they may be different by the time the team comes to build them. Such an approach makes it more difficult to estimate the completion time of a project, but it should result in less time wasted on planning.

The second different in agile planning is that planning of iterations is done on the requirements level, rather than on individual tasks such as designing and implementing. That is, the plan for an iteration would have tasks such as "Complete requirement X", rather than "Design", "Implement", and "Test".

General opinion is that tools such as Gantt charts seem and PERT charts have little use in agile planning, even at the iteration level. Instead, simply having a list of the tasks that must be completed is highly valuable, and relatively lightweight. One aspect in which Gantt charts are useful is to track dependencies between tasks. Some practitioners see the value in using these high-level aspects of Gantt charts, but calling these "Gantt charts" is really a stretch. Instead, they are most likely using some planning software with Gantt charts to have tasks lists with dependencies.

The following is a list of rules that are typically used in agile planning:

- *Plan short iterations*: Using shorter iterations give the team a measurable progress indicator. Functionality is delivered on a regular basis.
- *Produce useful functionality*: Each iteration should produce a useful piece of functionality; for example, one requirement.
- *Using "Just in time" (JIT) planning*: Only plan for iterations that will be starting in the near future.
- *Use the team*: All members of the team should be involved in both the scheduling and the assignment of word to team members. Not only does this promote a sense of belonging, but the team will have to implement the plan, so are motivated to get it right. Furthermore, they know their strengths and preferences, so involving them in the planning ensures that they do the work they want — within reason!.

One of my favourite agile planning tips comes from Scott Ambler of Ambysoft[2], as it demonstrates the weaknesses of people as resources:

> "*Ignore the calendar*. If your iteration is one week in length, run it from Wednesday to Tuesday. Iterations which run Monday to Friday typically suffer from "weekend-itis" where your focus and energy is reduced dramatically on Friday because it's the end of the iteration and the end of the week. This is effectively a double whammy to your productivity."

Dangers of agile planning propaganda

One of the strengths of agile development over more formal development methods is that it better prepares teams for change by not having fixed plans. However, like any movement, there are extremists whose opinions on the subjects can be dangerous if taken on board.

For example, I have seen it written and heard it said that traditional methods of development are useful for nothing, and that agile projects should be used for everything. Furthermore, in these agile projects, you should *never* plan. This type of propaganda is nothing short of insane, and the people who champion it could not have possibly built a large-scale system.

While agile projects work effectively on small-to-medium scale systems, such as small desktop applications or web-sites with database backends, assuming that the same techniques will work for embedded systems that control aircraft missile systems or air traffic is off the mark. For systems in which safety, reliability, and security are important

[2]See `http://www.ambysoft.com/essays/agileProjectPlanning.html`.

factors, an agile approach is less likely to be effective, because the requirements of these products really do need to be carefully planned, as to the projects that build them.

Fortunately, most agile development proponents take a balanced stance, and appreciate the shortcomings of agile planning, as well as the strengths.

Agile projects are also less suitable for teams who are geographically separated. Agile methodologies typically promote teams working closely together on small tasks, and aim to minimise documentation. However, for a team that is separated over different locations, this model does not work as effectively. While techniques have been proposed to mitigate this problem, such as getting agile teams together for 1-2 months at the start of the project, these are costly overheads for some organisations.

References

[Bro95] F.P. Brooks. The mythical man-month. In *Essays on software engineering*. Addison-Wesley, 1995.

[Pre09] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw Hill, seventh edition, 2009.